JAVA 内部类分析与应用研究①

The Analysis of JAVA Inner class and Application

浦云明　（厦门 集美大学 361021）

摘要：本文分析了 JAVA 内部类，内部类的存储分配机制，说明了程序设计中如何巧妙地应用 JAVA 的各种内部类，内部类的应用可以简化程序设计，保证程序代码的清晰明了，提高程序运行效率。

关键词：JAVA　内部类　包含类

1　引言

内部类（inner class）又称为嵌入类（nested class），简单说就是在一个类中定义另一个类，这与在类中定义的变量和方法是类的成员一样，我们可以定义一个类（内部类）成为另一个类的成员。内部类是从 JDK1.1 版本增加的特性，它像块一样可以定义在程序的任何地方，开发过程中，内部类的使用可以确保程序清晰和精炼。

本质上说，内部类与其他类一样，它只是定义在其他类中间，内部类可以分为常规内部类，方法中定义内部类，静态内部类，匿名内部类，方法参数内部类。在本文中将对各种内部类进行分析并给出应用示例。

2　常规内部类 regular inner class

常规内部类（regular inner class）定义在一个类的内部，但必须在任何方法或代码块之外，它属于类的一部分。常规内部类可以使用修饰符 public、private、protected、abstract 和 final，但 abstract 和 final 这两个修饰符不能同时使用，因为 abstract 类必须有子类继承，而 final 类不能有子类继承。

内部类的实例与包含类的实例有特殊的关联，因此内部类可以访问所有的包含类的成员，包括 private 成员。对内部类实例化时，根据实例化在程序中位置，分为两种处理情况：

（1）当在包含类内部实例化内部类时，可以直接对内部类实例化如下，Inner 是内部类名称：

```
Inner in = new Inner( );
```

（2）当在包含类的外部实例化内部类时，应使用包含类名和内部类名，并使用包含类的一个引用/对象/实例来实例化如下，Outer 和 Inner 分别是包含类和内部类名称，out 是包含类的一个引用，in 是内部类的引用：

```
Outer out = new Outer( );
Outer. Inner in = out. new Inner( );
```

当然，也可以把上边两行连接起来，更简洁的格式对内部类实例化：

```
Outer. Inner in = new Outer( ). new Inner( );
```

编译器编译常规内部类时，会创建一个名称为 EnclosingClassName $ Innerclassname. class 的文件，其中 EnclosingClassName 是包含类的类名，Innerclassname 是内部类名，如例子 1 中，包含类名是 Outer，内部类名是 Inner，编译后会生成名为 Outer $ Inner. class 的文件。

在内部类代码中的关键字 this 是内部类的一个引用，若需要包含类的一个引用，需要在关键字 this 前加上包含类名，如 Outer. this。

例子 1：

```
class Outer{
  private int x = 3;
  public void makeinner( ){
    Inner in = new Inner( ); //本例子中，包含类
```

① 本课题得到“福建省教育厅科技项目资助”

的方法,对内部类进行了实例化

```
        in. seeouter( ) ;
    }
    class Inner{
        public void seeouter( ) {
            System. out. println( " outer x is: "  + x) ;
            System. out. println( " inner class ref is "  +
this) ; //this 是内部类一个引用
            System. out. println( " outer class ref is "  +
Outer. this) ; //Outer. this 是包含类一个引用
        }
    }
    public static void main( String[ ] args) {
        Outer. Inner in  = new Outer( ). new Inner
( ) ; //直接对内部类实例化,并调用内部类方法
        in. seeouter( ) ;
        / *
```

本例中,由于包含类的方法 makeinner( ) 中对内部类进行了实例化,因此,我们也可以用以下两行代码,对包含类实例化,调用包含类方法,进而实例化内部类并调用内部类方法

```
        Outer out  = new Outer( ) ;
        out. makeinner( ) ; * /
    }
}
```

程序运行结果:

```
outer x is:3
inner class ref is Outer $ Inner@126b249
outer class ref is Outer@182f0db
```

3　方法中定义的内部类 method – local inner class

如果在包含类的一个方法中定义一个类,我们称这个内部类是方法中定义的内部类 method local inner class。对方法中定义的内部类实例化,必须在定义的方法中,而且必须在内部类定义后对其实例化。

内部类不能使用方法中的变量,除非该变量有 final 修饰符(也就是常量)。这是因为方法中定义的变量是局部的,它分配在堆栈中,当方法的生命期结束时,方法中变量的生存期也就结束了,但方法中定义的内部类的实例可能还在内存堆中存活。例如,它的一个引用被传到了其他代码中,因而方法变量的生存期与方法中定义的内部类实例的生存期不能保证一致,所以除非是 final 修饰符的变量,不能使用方法中的变量。

与常规内部类不同,public、private、protected、static、transient 不能修饰方法中定义的内部类,可用的修饰符只能是 abstract 和 final,而且这两个修饰符不能同时使用。

编译器编译方法中定义的内部类时,会创建一个名称为 EnclosingClassName $ n $ Innerclassname. class 的文件,其中 EnclosingClassName 是包含类的类名,n 是一个编译器生成的整数,Innerclassname 是内部类名,例如下例中编译后会生成名为 Outer $ 1 $ Inner. class 的文件。如例子 2 中,包含类名是 Outer,内部类名是 Inner,编译后会生成名为 Outer $ 1 $ Inner. class 的文件。

例子 2:

```
class Outer{
    private int x = 3;
    public void dostuff( ) {
        final int y =  9;
        class Inner{
            public void seeouter( ) {
                System. out. println( " outer x is: "  + x
+ " method final y is: "  + y) ; //y 必须是 fianl 修饰
符,即常量
                System. out. println( " inner class ref is"
+ this) ;
                System. out. println( " outer class ref is"
+ Outer. this) ; }
        }
        Inner in  = new Inner( ) ; //注意,实例化语
句必须在内部类定义后
        in. seeouter( ) ; //调用内部类 Inner 的方法
seeouter( )
    }
    public static void main( String[ ]args) {
        Outer out  = new Outer( ) ; //another way is
like above 2 line
        out. dostuff( ) ; }
}
```

程序运行结果:

```
outer x is:3 method final y is :9
inner class ref is Outer$1$Inner@126b249
outer class ref is Outer@182f0db
```

4 静态内部类 static inner class

在讨论静态内部类前,我们先来说明一下类中的静态方法的特性:

(1) 静态方法包括 main(),不能使用 this 引用;

(2) 静态方法不能调用对象变量,只能调用静态变量;

(3) 静态方法不能被覆盖(overriding,继承的时候);

(4) 静态方法可以直接调用静态方法,但不能直接调用非静态方法,除非使用对象名.方法名。

使用了修饰符 static 的内部类,我们称之为静态内部类。严格的技术意义上讲,静态内部类不应算作内部类,但一般将它看作顶层的内部类(top – level inner class)。静态内部类与包含类(outer class)对象/实例没有实际的联系,因此静态内部类不能访问包含类中的非静态变量/对象变量。

实例化静态内部类时,与常规内部类实例化不同,不能使用包含类的对象来实例化内部类,也就是说内部类对象不能使用包含类的引用。实例化内部类时,需要包含类和内部类,格式如下:

```
Outerclassname.Innerclassname in = new Outerclassname.Innerclassname();
```

编译器编译静态内部类时,会创建一个名称为 EnclosingClassName $ Innerclassname.class 的文件,如例子 3 中,包含类名是 Outer,静态内部类名是 Inner,编译后会生成名为 Outer $ Inner.class 的文件。

例子 3 中,静态内部类的方法 seeouter() 可以访问包含类的静态变量 x(必须是静态),不能访问包含类的非静态变量,不能使用包含类的引用。

例子 3:

```
class Outer{
  static int x=3;
  static class Inner{ //注意:静态内部类中不能访问非静态成员
    public void seeouter(){
      System.out.println("outer x is: " + x);
//在静态内部中只能访问静态变量
      System.out.println("inner class ref is" + this); //输出内部类的一个引用
      //System.out.println("outer class ref is" + Outer.this); //出错,因为内部类实例不能使用包含类的引用
    }
  }
  public static void main(String[]args){
    Outer.Inner in = new Outer.Inner(); //static nested class //实例化静态内部类
    in.seeouter(); //访问内部类方法 seeouter
  }
}
```

程序运行结果:

```
Outer x is:3
Inner class ref is Outer$Inner@126b249
```

5 匿名内部类 anonymous inner class

匿名内部类是没名字的内部类,一般可分为 2 种形式:常规定义的匿名内部类,作为方法的参数定义的匿名内部类。

5.1 常规定义的匿名内部类

匿名内部类没有名字,匿名类可以继承另一类,或实现一个接口,语法上来说,这两者不能同时实现,JAVA 也不允许实现多接口(唯一的例外是一个类继承另一个类,而该类的父类实现一个接口)。匿名内部类不能显式定义构造器。匿名内部类的声明和使用采用如下格式:

```
Classname AnClassname = new Classname(){ /* class body */ };
```

其中,Classname 是类或接口名,一定要明确代码是一个表达式,它返回一个对象。可以使用在对象引用的任何地方。

匿名内部类是语句的一部分,因而在大括号后必须紧跟分号。这一点在 JAVA 程序中也是少见的。编译器编译匿名内部类时,会创建一个独立的名称为 EnclosingClassName $ n.class 的文件,其中 EnclosingClassName 是包含匿名内部类的类名,n 是一个编译器生成的整数。在例 4 中,编译后会生成 Outer $ 1.

class。而原先定义的类 Inner 还是存在的,可以对类 Inner 实例化,调用 dostuff( )方法。参见例子 4 的运行结果。

例子 4:

```
class Inner{
  private int x = 7;
  public void doStuff( ){
    System. out. println ( " inner class x is "  +
x) ;}
}
class Outer{
  private int x = 3;
  Inner in  =  new Inner( ){
    public void doStuff( ){
      System. out. println ( " anonymous inner
class x is: "  + x) ; //包含类 Outer 的成员 x
      System. out. println ( " anonymous inner
class ref is "  + this) ; //匿名内部类的引用
      System. out. println( " outer class ref is "  +
Outer. this) ; //包含类的引用
    }
  } ; //不能忘记分号
}
class Outer2{
  public static void main( String[ ] args) {
    Outer out  =  new Outer( ) ;
    out. in. doStuff( ) ;
    Inner in  =  new Inner( ) ;
    in. dostuff( ) ;}
}
```

程序运行结果:

```
anonymous inner class x is: 3
anonymous inner class ref is Outer $ 1@ 26b249
outer class ref is outer@ 82f0db
inner class x is: 7
```

5.2 方法中参数定义匿名内部类 argument – defined anonymous inner class

对于方法中定义的匿名内部类,由于不知道匿名类的名字,因此不能使用 new 关键字来创建类的实例,声明和构造匿名内部类时,接口名或类名直接紧跟在 new 后,一方面,该句法定义一个类,但不需要类的类型,另一方面,匿名类实现了一个接口,或继承另一类,而不需要使用 implements 或 extends 关键字。这种匿名类的构造很方便,但不能在其他地方对该类实例化,因为只有在匿名类代码出现的地方才能实例化。另外,匿名类必须是代码较少,如果类的方法中,若匿名类前有多于 2 行的代码,或者匿名类有超过 10 行的代码,就不应考虑使用匿名类了。

对于方法中参数定义的匿名内部类,匿名类的定义、构造和使用发生在同一地方同时实现。也就是说匿名内部类必须包括在方法中。同样地方法中定义的匿名内部类不能显式定义构造器。例子 6 中,在方法 addActionListener( )中,参数定义了一个匿名内部类,它实现一个接口 ActionListener,代码如下:

例子 6:

```
public void aMethod( ){
  theButton. addActionListener(
    new ActionListener( ){ //开始声明和构造匿
名内部类,接口名直接紧跟在 new 后
      public void actionPerformed( ActionEvent e)
{
        System. out. println( " the action has oc-
cured" ) ;
      }
    }
  ) ; //注意这里的分号不能少
}
```

6 小结

本文中讨论了各内部类并给出应用示例,内部类的使用可以确保程序清晰和精炼。本文的例子均已在 J2sdk1. 4. 0 环境下调试通过,在教学和培训中收到了很好的效果。

参考文献

1 Java 2 认证考试学习指南,(美)罗伯特等著,电子工业出版社,2000 年 11 月。

2 Sun Certified Programmer & Developer for Java 2,人民邮电出版社,2004 年 2 月。

3 JAVA How to Program, 4th edition, Harvey M. Deitel,电子工业出版社,2002 年 6 月。